Facultad de Ciencias - UNAM
Estructuras Discretas 2026-1
Práctica 2: Listas

Favio Ezequiel Miranda Perea Patricio Ordoñez Blanco Eduardo Vargas Pérez

26 de Septiembre de 2025
Fecha de entrega: 3 de Octubre de 2025 hasta las 23:59

Desarrollo de la práctica

Binarios

Tanto en el laboratorio como en las clases de teoría se trabajó con cadenas binarias representadas al revés. Esto es, $6_{10} = 110_2$ en realidad lo representamos como 011_2.

Su trabajo para esta parte es modificar las funciones que se realizaron para que la representación de los números binarios ya no sea al revés. Para ello, deben considerar el siguiente tipo de dato, así como el siguiente sinónimo:

```
data Bit = O | I
type Binario = [Bit]
```

Dado lo anterior, deben implementar las siguientes funciones.

1. `toDecimal :: Binario -> Int`
   Dado un número binario, esta función debe calcular el valor decimal de ese número. **[1 Punto]**
   Ejemplos

   ```
   ghci > toDecimal [I,O,O,O] ▷ 8
   ghci > toDecimal [I,I,O,I,O] ▷ 26
   ```

2. `toBin :: Int -> Binario`
   Dados número en representación decimal, esta función debe devolver la representación binaria de ese número. **[1 Punto]**
   Ejemplos

   ```
   ghci > toBin 26 ▷ [I,I,O,I,O]
   ghci > toBin 54 ▷ [I,I,O,I,I,O]
   ```

3. `suma :: Binario -> Binario -> Binario`
   La función recibe dos números binarios y debe regresar la suma de los mismos. **[1 Punto]**
   Ejemplo

```
ghci > suma [I,O,I,O,O] [I,O,I] ▷ [I,I,O,O,I]
```

Listas

Realizar la implementación de las siguientes funciones

1. `palindromo :: [a] -> Bool`
   Dada una lista, esta función tiene que devolver `True` si la lista es un palíndromo, `False` en otro caso. **[0.5 Puntos]**
   Ejemplos

```
ghci > palindromo "ana" ▷ True
ghci > palindromo [1,2,3,4] ▷ False
```

   Nota: Pueden asumir que no hay acentos o espacios si se llega a pasar una cadena `String`.

2. `diferenciaSimetrica :: [a] -> [a] -> [a]`
   Dadas dos listas, esta función debe devolver el resultado de aplicar la diferencia simétrica entre las dos listas. Es decir, los elementos que estén en la unión, pero no en la intersección.
   [1 Punto]
   Ejemplo

```
ghci > diferenciaSimetrica [1,2,3,4] [3,4,5] ▷ [1,2,5]
```

3. `conjuntoPotencia :: [a] -> [[a]]`
   Dada una lista, esta función debe calcular el conjunto potencia de la lista ingresada.
   [1.5 Puntos]
   Ejemplo

```
ghci > conjuntoPotencia [1,2] ▷ [[1,2],[1],[2],[]]
```

   Hint: Usa recursión y listas por comprensión.

Listas de longitud par

Para garantizar una representación de listas de longitud par, deben proporcionar algún sinónimo llamado `ListaPar a b` que sea una lista de pares ordenados. **[0.5 Puntos]**

1. `longitud :: ListaPar a b -> Int`
   La función debe recibir una lista de pares y devolver la longitud. **[0.5 Puntos]**
   Ejemplo

```
ghci > longitud [(1,10), (2,20), (3,30)] ▷ 6
```

2. `myMap :: (a -> c) -> (b -> d) -> ListaPar a b -> ListaPar c d`
   Investigar la función `map` y trasladar la idea sobre las listas definidas en esta sección. **[1 Punto]**
   Ejemplos

```
ghci > myMap (*2) (*3) [(1,2), (3,4), (5,6)] ▷ [(2,6),(6,12),(10,18)]
```

3. `sumaPares :: ListaPar a b -> (a,b)`
   La función debe recibir una lista de pares y devolver una tupla. La primer entrada de la tupla debe ser el resultado de sumar la primer entrada de cada par de la lista, la segunda entrada debe ser el resultado de sumar la segunda entrada de todos los pares de la lista. **[1 Punto]**
   Ejemplo

   ```
   ghci > sumaPares [(1,10),(2,20),(3,30)] ▷ (6,60)
   ```

4. `myFilter :: ((a,b) -> Bool) -> ListaPar a b -> ListaPar a b`
   Investigar la función `filter` y trasladar la idea sobre las listas definidas en esta sección.
   [1 Punto]
   Ejemplo

   ```
   ghci > myFilter (\(x,y) -> even x && even y) [(1,2),(2,4),(3,5),(6,8)]
                                ▷ [(2,4),(6,8)]
   ```

Limitaciones

Recuerden que toda función auxiliar que necesiten (con excepción de las funciones *div* y *mod*) tiene que ser implementada por ustedes.

Notas

La firma de algunas funciones no está completa, es necesario asegurarse de que algunos tipos cumplan con ser parte de alguna clase de tipo para poder realizar alguna otra función.

¡Buena suerte a todos! ☺☺☺